DETAILED ACTION

1. Claims 1-24 are pending.

2. All claims are rejected.

Response to Arguments

3. Applicant's arguments, see pg. 3, filed March 10, 2008, with respect to the specification, have been fully considered and are persuasive in light of the amendments. The objection to the specification has been withdrawn.

4. Applicant's arguments, see pp. 4-5, filed March 10, 2008, with respect to claims 10-12, have been fully considered and are persuasive. The rejection under 35 U.S.C. § 112 ¶ 1 of claims 10-12 has been withdrawn.

Applicant's arguments, see pp. 5-9, filed March 10, 2008, with respect to claims 1-24, have been fully considered and are persuasive. The rejection under 35 U.S.C. § 103(a) of claims 1-22 has been withdrawn. However, upon further consideration, a new ground(s) of rejection is made in view of Long and Kuhns.

Claim Rejections - 35 U.S.C. § 103

5. The text of those sections of Title 35, U.S. Code not included in this action can be found in a prior Office action.

6. Claims 1, 23 and 24 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of

Kuhns, *Core Inter-Process Communication Mechanisms*.

As per claim 1, Long teaches:

initiating a process-wide deactivation operation (Long, ¶ 0008, "In a multi-threaded environment, a 'stop' instruction is typically sent to all threads prior to performing a global operation."), where the sending is the operation as claimed;

determining whether threads of the process are currently suspendable (Long, ¶ 0008, "Once a stop instruction is sent to all threads, it is generally determined whether each thread is in a 'safe' region or an 'unsafe' region.");

moving the threads of the process that are currently suspendable to a stopped state (Long, ¶ 0008, "Conventionally, all threads are typically suspended in order to evaluate each thread and to determine the disposition of each individual thread, i.e., to determine whether each thread is in a safe region or an unsafe region. When a thread is determined to be in an unsafe region, the thread is typically allowed to resume its operation, and is stopped at a subsequent point in time in order to attempt to suspend the thread at a safe region."), where it is clear that currently-safe threads are being suspended;

Long does not specifically teach:

the process-wide deactivation operation is called by outstanding threads of the process when the outstanding threads re-enter a kernel of the operating system.

The analogous and compatible art of Kuhns, however, does (Kuhns, pg. 7, "if pending signal and processes has handler, then kernel arranges to first run handler on returning to user mode, then resume the interrupted instruction.").

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teaching of Kuhns with the teaching of Long to provide a means of integrating the safe suspend method of Long with the operating system kernel.

Claims 23 and 24 correspond to claim 1 and are rejected under the same reasons set forth with claim 1 above.

7. Claim 2 is rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of Kuhns, *Core Inter-Process Communication Mechanisms*, Toutonghi et al., U.S. Pat. No. 5,842,016 (hereinafter "Toutonghi") and Nemirovsky et al., U.S. Pat. No. 7,020,879 (hereinafter "Nemirovsky").

As per claim 2, the rejection of claim 1 is incorporated, but the incorporated teachinges do not teach:

a thread is currently suspendable if the thread is stopped.

However, the analogous and compatible art of Toutonghi teaches that a thread that is suspendable is one that is "at a stable point" (Col. 16, lines 14-17). Further, Nemirovsky teaches that a thread that is not executing is at a stable point (Col. 18, lines 11-14).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Toutonghi and Nemirovsky into the teachings of Long and Kuhns as providing a criteria by which threads may be determined suspendable.

8. Claim 3 is rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of Kuhns, *Core Inter-Process Communication Mechanisms*, Toutonghi et al., U.S. Pat. No. 5,842,016 (hereinafter "Toutonghi"), Nemirovsky et al., U.S. Pat. No. 7,020,879 (hereinafter "Nemirovsky") and Hogle et al., U.S. Pat. No. 6,560,626 (hereinafter "Hogle").

As per claim 3, the rejection of claim 2 is incorporated, but the incorporated teachinges do not teach:

a thread is currently suspendable if the thread is sleeping interruptibly.

However, the analogous and compatible art of Hogle teaches that a thread that is

interruptibly sleeping is not executing (Col. 1, lines 46-50).

It would have been obvious to one of ordinary skill in the art at the time the invention was made that the interruptibly sleeping thread taught by Hogle is at a stable point as taught by Nemirovsky, and is therefore suspendable as taught by Toutonghi.

9. Claim 4 is rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of Kuhns, *Core Inter-Process Communication Mechanisms*, Toutonghi et al., U.S. Pat. No. 5,842,016 (hereinafter "Toutonghi"), Nemirovsky et al., U.S. Pat. No. 7,020,879 (hereinafter "Nemirovsky"), Hogle et al., U.S. Pat. No. 6,560,626 (hereinafter "Hogle") and Hsieh, U.S. Pat. No. 7,210,146 (hereinafter "Hsieh").

As per claim 4, the rejection of claim 3 is incorporated, but the incorporated teachinges do not teach:

a thread is also currently suspendable if the thread is interruptible and not currently running; and

the thread is removed from a run queue prior to moving the thread to the stopped state.

However, the analogous and compatible art of Hseih teaches that an interruptible thread can be put to sleep (Col. 4, lines 53-55), that such a state is "non-executing"

(Col. 1, lines 48-50), and that putting a thread to sleep can be accomplished by removing it from the run queue before putting it in a stopped state on a sleep queue (Col. 1, lines 58-60; col. 2, lines 30-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made that the interruptible thread of Hsieh is at a stable point as taught by Nemirovsky, and is therefore suspendable as taught by Toutonghi, and that the thread can only be in one place at a time, so it should be removed from the run queue before being placed anywhere else.

10. Claims 5-10 and 12 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of Kuhns, *Core Inter-Process Communication Mechanisms* and Riel, *Linux 2.4.5-ac5-swapper Patch* (hereinafter "Riel").

As per claim 5, the rejection of claim 1 is incorporated, but the incorporated teachinges do not teach:

determining whether threads of the process are sleeping on memory; **and**

counting the threads of the process that are sleeping on memory to determine a number of memory sleepers.

However, the analogous and compatible art of the Swapper Patch does. It

teaches a method of determining whether threads of the process are sleeping on memory (Swapper Patch, "waitqueue_active(&memory_queue)"), and that these threads are kept in a queue. It is well-known in the art how to count the number of entries in a queue.

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teaching of Riel with the teachings of Long and Kuhns so as to take advantage of the memory management method taught by Riel that requires a determination as claimed.

Long teaches counting as a means to determine whether or not the process as a whole is in a condition to be suspended (Long, ¶ 0042, "When it is determined that the inconsistent count is not greater than zero, the indication is that substantially all threads in a multi-threaded system are consistent.").

It would have been obvious to one of ordinary skill in the art at the time the invention was made that stopping a memory sleeping thread will leave the system in a state that is no more unsafe than it was prior to stopping the thread. Memory sleepers are placed on a sleep queue when there is insufficient memory to run them. The need to ensure the safety of process deactivation is known in the art. It would have been obvious to one of ordinary skill in the art at the time the invention was made therefore to treat them as if they were consistent for the purposes of process deactivation.

However, the method of Long, to compare the number of inconsistent threads to zero, would need to be modified in order to take this realization into account by comparing the number of consistent threads and memory sleepers to total threads to determine if the process as a whole can be safely deactivated.

As per claim 6, the rejection of claim 5 is incorporated, and further Long teaches:

counting as a means to determine whether or not the process as a whole is in a condition to be suspended (Long, ¶ 0042, "When it is determined that the inconsistent count is not greater than zero, the indication is that substantially all threads in a multi-threaded system are consistent.").

As per claim 7, the rejection of claim 6 is incorporated and further Long teaches:

counting as a means to determine whether or not the process as a whole is in a condition to be suspended (Long, ¶ 0042, "When it is determined that the inconsistent count is not greater than zero, the indication is that substantially all threads in a multi-threaded system are consistent.").

As per claim 8, the rejection of claim 7 is incorporated and further Long teaches:

returning and indicating self-deactivation pending if the sum is unequal to

the number of live threads (Long, ¶ 0042, "After the wait on the consistent state condition variable is completed, i.e., after the requesting thread receives a notification, then process flow returns to step 430 in which a determination is made as to whether the inconsistent count is still greater than zero."; Long, Figure 4).

As per claim 9, the rejection of claim 7 is hereby incorporated, and further Long teaches:

moving the threads of the process that are suspendable to a stopped state. (Long, ¶ 0008, "Conventionally, all threads are typically suspended in order to evaluate each thread and to determine the disposition of each individual thread, i.e., to determine whether each thread is in a safe region or an unsafe region. When a thread is determined to be in an unsafe region, the thread is typically allowed to resume its operation, and is stopped at a subsequent point in time in order to attempt to suspend the thread at a safe region.").

It would have been obvious to one of ordinary skill in the art at the time the invention was made that memory sleepers are suspendable; the fact that they are memory *sleepers* signifies that they are not currently running, and if they are not running, their continuing to not run cannot negatively impact the stability of the system.

As per claim 10, the rejection of claim 9 is hereby incorporated, and further Long teaches:

a method of determining whether a process is deactivatable, namely, whether or not there are any unsafe or inconsistent threads (Long, ¶ 0042, "After the wait on the consistent state condition variable is completed, i.e., after the requesting thread receives a notification, then process flow returns to step 430 in which a determination is made as to whether the inconsistent count is still greater than zero."; Long, Figure 4).

As per claim 12, the rejection of claim 10 is hereby incorporated, and further Long teaches that:

a safe deactivation process is a global operation (¶ 0006, "While maintaining multiple threads, a computing system may need to perform global operations which require synchronization, or control of all threads or a set of threads at a given time").

If the process cannot deactivate due to an inconsistent thread in the process, the global operation cannot be completed. As the process cannot deactivate, and the prior steps involve moving threads to the stopped state, i.e. suspending them, it would have been obvious to one of ordinary skill in the art at the time the invention was made to un-suspend them so as to leave the process in the same state that it was in prior to the initialization of the process-wide deactivation procedure.

11. Claim 11 is rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of Kuhns, *Core Inter-*

Process Communication Mechanisms and Riel, *Linux 2.4.5-ac5-swapper Patch* (hereinafter "Riel") and further in view of Cawley, U.S. Patent No. 5,361,334 (hereinafter "Cawley").

As per claim 11, the rejection of claim 10 is hereby incorporated, but the incorporated teachings do not teach:

a flag may be set to signal the completion of the process

Cawley teaches the use of flags to indicate that a process is runnable. (Col. 5, lines 5-8).

It would have been obvious to one of ordinary skill in the art at the time the invention was made that setting the runnable flag to false would signal the process scheduler not to schedule the process to execute on the processor.

12. Claim 13 is rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of Kuhns, *Core Inter-Process Communication Mechanisms* and further in view of Rogers et al., U.S. Patent No. 5,557,747 (hereinafter "Rogers").

As per claim 13, the rejection of claim 1 is hereby incorporated, but the incorporated teachinges do not teach:

not analyzing zombie threads.

The analogous art of Rogers teaches that a zombie thread is one that "has been terminated." (Col. 10, lines 52-53).

It would have been obvious to one of ordinary skill in the art that a terminated thread would not affect whether or not the process to be deactivated, and therefore could safely be ignored and not analyzed.

13. Claim 14 is rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of Kuhns, *Core Inter-Process Communication Mechanisms* and further in view of Davy, U.S. Patent No. 5,517,643 (hereinafter "Davy").

As per claim 14, the rejection of claim 1 is hereby incorporated, but the incorporated teachinges do not teach:

deactivating a process using a memory swapper of an operating system.

The analogous art Davy, however, teaches the use of a "SWAPPER" that acts to free memory by swapping out pages. (Col. 4, lines 10-15).

It would have been obvious to one of ordinary skill in the art to have the memory

swapper implement the method described in the application, as the memory swapper described in Davy is designed to free memory.

14. Claims 15, 16 and 21 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of Kuhns, *Core Inter-Process Communication Mechanisms* and further in view of Farrell et al., U.S. Patent No. 5,247,675 (hereinafter "Farrell").

As per claim 15, the rejection of claim 1 is hereby incorporated, but the incorporated teachinges do not teach:

compelling a thread re-entering the kernel to enter the process deactivation operation.

The analogous art of Farrell, however, shows that the operating system may force threads to undertake certain actions. (Col. 10, lines 22-24; Col. 10, lines 49-63).

It would have been obvious to one of ordinary skill in the art to apply the teaching of Farrell because of the benefit of having a managed process deactivation system.

As per claim 16, the rejection of claim 15 is hereby incorporated, and further Farrell teaches:

the re-entered thread removing itself from its process. (Col. 10, lines 37-

40).

It would therefore have been obvious to one of ordinary skill in the art that a re-entering thread could manage its own running state, and move itself to the stopped state if it were compelled to enter the kernel by virtue of a process-wide deactivation operation.

As per claim 21, the rejection of claim 1 is hereby incorporated, and further Farrell teaches:

threads communicate with the kernel. (Col. 10, lines 26-28).

It would have been obvious to one of ordinary skill in the art that a thread going to sleep would communicate with the kernel that information, and that this information would include whether the process is being deactivated such that the thread should not be woken up until the operating system determines that the process is to be unsuspended.

15. Claims 17, 19 and 20 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of Kuhns, *Core Inter-Process Communication Mechanisms* and further in view of Mounes-Toussi et al., U.S. Patent No. 6,269,425 (hereinafter "Mounes-Toussi").

As per claim 17, the rejection of claim 16 is hereby incorporated, and further neither Kawahara nor Farrell teach:

counting the number of threads that are sleeping on memory.

The analogous art of Mounes-Toussi, however, does. (Col. 3, lines 50-52)

It would have been obvious to one of ordinary skill in the art to apply this analogous art as keeping track of the number of threads sleeping on memory is of obvious utility in a memory management system. As well, neither Kawahara, Farrell nor Mounes-Toussi disclose calculating a sum of a number of threads in the stopped state and the number of threads sleeping on memory.

The analogous art of Long teaches that one can determine a number of threads that are in an "unsafe" condition. (¶ 0040).

Having stopped suspendable threads, it would be obvious to one of ordinary skill in the art that both stopped threads and memory sleepers cannot execute or otherwise use any resources of the computer while remaining in their current state. It would have been obvious to one of ordinary skill in the art that one could alternatively count the number of threads that are in a safe condition – that is, the sum of those threads that are stopped as a result of having been determined to be suspendable and those threads that are memory sleepers. As well, neither Kawahara, Farrell nor Mounes-

Toussi disclose determining if the sum of the number of memory sleepers and the number of stopped threads.

The analogous art of Long, however, teaches that a determination can be made if there are any unsafe threads by comparing the number of unsafe threads to zero. (¶ 0041).

It would have been obvious to one of ordinary skill in the art that determining whether the number of unsafe threads is zero is functionally equivalent to determining if the number of safe threads is equal to the number of threads.

As per claim 19, the rejection of claim 17 is hereby incorporated, and further Long teaches:

a method of determining whether a process is deactivatable, namely, whether or not there are any unsafe or inconsistent threads. (Long, ¶ 0042, "After the wait on the consistent state condition variable is completed, i.e., after the requesting thread receives a notification, then process flow returns to step 430 in which a determination is made as to whether the inconsistent count is still greater than zero."; Long, Figure 4).

It would have been obvious to one of ordinary skill in the art that actually determining if the process is deactivatable is a necessary step before deactivating the

process, and further that if the process is not deactivatable, then those steps that were undertaken in furtherance of deactivating the process ought to be undone.

As per claim 20, the rejection of claim 17 is hereby incorporated, and further Long teaches:

a method of determining whether a process is deactivatable, namely, whether or not there are any unsafe or inconsistent threads. (Long, ¶ 0042, "After the wait on the consistent state condition variable is completed, i.e., after the requesting thread receives a notification, then process flow returns to step 430 in which a determination is made as to whether the inconsistent count is still greater than zero."; Long, Figure 4).

It would have been obvious to one of ordinary skill in the art that if the process is determined to be deactivatable and all threads of the process are suspended, stopped or sleeping, then the process is completed.

16. Claim 18 is rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of Kuhns, *Core Inter-Process Communication Mechanisms* and further in view of Klemm et al., U.S. Patent No. 7,243,267 (hereinafter "Klemm").

As per claim 18, the rejection of claim 17 is hereby incorporated, but the

incorporated teachings do not teach:

un-suspending threads of the process and ending the process-wide deactivation operation if a kill command has been received.

Klemm teaches that a kill command is used to signal to the operating system that a particular process is to be forcibly terminated. (Col. 2, lines 57-59).

It would have been obvious to one of ordinary skill in the art that the reception of the kill signal would indicate that the process was to be terminated, and that unsuspending the threads of the process would permit the termination and later reacquisition of the allocated memory of the threads, and also that allowing the threads to remain in a suspended state would only delay this eventual step.

17. Claim 22 is rejected under 35 U.S.C. § 103(a) as being unpatentable over Long et al., U.S. PGPub 2001/0054057 (hereinafter "Long") in view of Kuhns, *Core Inter-Process Communication Mechanisms* and further in view of Davy, U.S. Patent No. 5,517,643 (hereinafter "Davy") and Cawley, U.S. Patent No. 5,361,334 (hereinafter "Cawley").

As per claim 22, the rejection of claim 1 is hereby incorporated, but the incorporated teachings do not teach:

reactivating a deactivated process by a procedure that includes bringing in

associated memory regions, turning on a flag, and un-suspending threads of the process.

The analogous art Davy, however, teaches that reactivating a process will generate so-called "hard faults." (Col. 4, lines 48-50). These hard faults are generated when the associated memory regions are being brought in when the threads of the process attempt to execute after having been un-suspended.

Cawley further teaches the use of flags to indicate that a process is runnable. (Col. 5, lines 5-8).

It would have been obvious to one of ordinary skill in the art that a thread whose pages were swapped to secondary storage would necessarily hard fault upon its execution, bringing in associated memory regions, while a flag can be set to indicate that the process is runnable, which would assist in process management.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to WILLIAM SPIELER whose telephone number is (571) 270-3883. The examiner can normally be reached on Monday to Thursday, 11 AM - 1 PM Eastern.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Trujillo James can be reached on (571) 272-3677. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/William Spieler/
Patent Examiner, AU 2169

/James K. Trujillo/
Supervisory Patent Examiner, Art Unit 2169